27/09/2023
Wednesday

* LEETCODE -> 2325 (Decode the message)

I/P = String key = "Data structure"

String message = "urrs"

O/P = feed

Substitution table →

Key →	d	a	t	a	_	s	t	r	u	c	t	u	r	e
	a	b	c	b	_	d	c	e	f	g	c	f	e	h

Message →	u	r	r	s
	f	e	e	d

Approach → step 1 - create mapping
step 2 - use mapping

// create mapping

- Initialise a variable start with a.
- Create a char array of any size greater than 256 and initialise it with 0. It is initialised with 0 so as one character is mapped only one time.
- Traverse a loop on string key.
- If → • character is not space && mapping[ch] == 0
  • Update mapping[ch] = start
  • Increment start
- ch represents the characters present in key string.

// use mapping,

= Create a string ans.
= Traverse a loop on string ~~ans~~ message.
= If → character is space
ans.push_back(' ');
else → ans.push_back(mapping[ch]);

= return ans;

Code→
```
#include<bits/stdc++.h>
using namespace std;

String decodeMessage(string &key, string &message){
    //create mapping
    char start='a';
    char mapping[300]={0};

    for(auto ch: key){
        if(ch!=' ' && mapping[ch]==0){
            mapping[ch]=start;
            start++;
        }
    }

    // use mapping,
    string ans;

    for(auto ch: message){
        if(ch==' '){
            ans.push_back(' ');
        else{
            ans.push_back(mapping[ch]);
    }}
```

```
    return ans;
}

int main(){
    string key = "data structure";
    string message = "urrs";
    string msg = decodeMessage(key, message);
    cout << msg;

}
```

- **Time complexity** → $O(m) + O(n) = O(m+n)$

$O(m)$ → key string
$O(n)$ → message string

- **Space complexity** → $O(1) + O(n) = O(n)$

$O(1)$ → constant space for char array mapping
$O(n)$ → ans string

__/__/____

* LEETCODE → 2391 (Minimum amount of time to collect garbage)

I/P = String garbage → {"MM", "PGM", "GP"}

Array travel → {5, 7}

O/P = total garbage collection time = 36

Garbage →

"MM"	"PGM"	"GP"
0	1	2

M: metal
P: Paper
G: Glass

Travel →

5	7
0	1

travel[i] is the time taken by truck to go from house i → i+1

i.e → 0 → 1 = 5
and 1 → 2 = 7

pickP	0	→	1	→	2	
pickG	0	→	1	→	2	
pickM	0	→	1	→	2	→ 3

lastP	0	→	1	→	2
lastG	0	→	1	→	2
lastM	0	→	1		

travelP	0	→ 5+7 = 12
travelG	0	→ 5+7 = 12
travelM	0	→ 5

total time = (pickP + travelP) + (pickG + travelG) + (pickM + travelM)
= (2 + 12) + (2 + 12) + (3 + 5) → 36

Approach →
Step 1 → Calculate pick time
Step 2 → Calculate travel time
Step 3 → Add pick time and travel time

Code →

```
#include<bits/stdc++.h>
using namespace std;

int garbageCollection(vector<string>&garbage, vector<int>&travel)
{
// These variables stores the time to pick garbage
// 1 unit to pick 1 garbage
    int pickG=0;
    int pickP=0;
    int pickM=0;

// These variables stores the last index where particular
// garbage found
// Jha tak position milegi specific garbage ki, vha
// tak us garbage k liye specific truck travel karenge
    int lastG=0;
    int lastP=0;
    int lastM=0;

// time taken by truck to travel from house i→i+1
    int travelG=0;
    int travelP=0;
    int travelM=0;
```

// calculate pick time

// garbage wali string par traverse karegi ye loop
```
for (int i=0; i<garbage.size(); i++){
```
// curr current index pr jo string h usse store karega
```
string curr= garbage[i];
```

// below loop har index pr jo string h uspe chlegi
```
for (int j=0; j<curr.length(); j++){
```
// ch current string k current character ko store karega
```
char ch= curr[j];

if (ch=='G'){
    pickG++;
```
// lastG ko i pe seta kiya taki pata chl jaye ki
// truck ko kis house i.e. index tak jana hai
```
    lastG=i;
}
if(ch=='P'){
    pickP++;
    lastP=i;
}
if(ch=='M'){
    pickM++;
    lastM=i;
}
}
}
```

// calculate travel time
```
for (int i=0; i<lastG; i++){
    travelG+=travel[i];
}
```

```
for (int i=0; i<lastP ; i++) {
        travelP += travel[i];
    }
for (int i=0; i<lastM; i++){
        travelM += travel[i];
    }
// total time = pick time + travel time

int totalTime = (pickG+travelG) + (pickP+travelP) +
                (pickM+travelM);
return totalTime;
}
int main() {
vector<string> garbage {"MM", "PGM", "GP"};
vector<int> travel {5, 7};

int finalAns= garbageCollection(garbage, travel);

cout<<"total garbage collection time = "<<finalAns;
}
```

Time complexity → O(n*m) + O(lastG) + O(lastP) + O(lastM)

Overall → O(n*m)

where, n → length of garbage vector

m → length of string at every index

Space complexity → O(1)

* Leetcode → 791 (Custom sort string.)

I/P = string s = "abcxyz"

string order = "yba"

O/P = ybacxz

Code →

```
#include<bits/stdc++.h>
using namespace std;

// str isliye bnaya so that ise order string k sath
// equate krke use kr sake custom comparator
// function mai

string str;

// string s se ch1 and ch2 my comp function k pas
// aayenge comparison k liye
// agr string str m bhi ch1 pehle aara hoga ch2
// se, then ye function true return kardega

// ye true return krega to string s m bhi ch1
// pehle aa jayega ch2 se

bool myComp(char ch1, char ch2){
    return (str.find(ch1) < str.find(ch2));
}
```

```
string customSort (string &s, string &order){
        str = order;
        sort(s.begin(), s.end(), myComp);
        return s;
    }

int main(){
        string s = "abcxyz";
        string order = "yba";

        string final = customSort(s, order);
        cout << final;
    }
```

Time complexity → O(nlogn) [∴ O(nlogn) → sort func.]

Space complexity → O(1)

__/__/__

* LEETCODE → 890 (Find and replace pattern)

I/P= words = {"abc", "deq", "mee", "aqq", "dkd", "ccc"}

pattern = "xyz"

O/P = abc deq

Approach→

words	Mapping of words
abc	abc
deq	abc
mee	abb
aqq	abb
dkd	aba
ccc	aaa

Pattern	Mapping of pattern
xyz	abc

- Compare string in words with pattern.
- If mapping of word == mapping of pattern
- Return that string in words for which above condition is true.

Code →

```
#include<bits/stdc++.h>
using namespace std;

void createUpdateMapping (string &str){

// create mapping
// this function will normalise the pattern and words
// in the similar format so that we can compare them
    char start = 'a';
    char mapping[300] = {0};
    for (auto ch: str){
        if (mapping[ch] == 0){
            mapping[ch] = start;
            start++;
        }
    }
// update the string
// ye words vector mai present har ek string
// k normalised format ko store krne k liye
// loop chalayi hai
    for (int i=0; i<str.length(); i++){
        char ch = str[i];
        str[i] = mapping[ch];
    }
}
```

__/__/____

```
vector<string> findSimilarPattern(vector<string>&words,
                                  string &pattern){
    vector<string> ans;
    // normalise the pattern
    createUpdateMapping(pattern);

    for(auto s: words){
        // store original string, in words vector in temps
        // agr store nahi karenge to output mai us string
        // ki mapping aa jayegi jo hume nhi caahiye
        string tempS= s;

        // normalise the words
        createUpdateMapping(tempS);
        if(tempS== pattern){
            ans.push_back(s);
        }
    }
return ans;
}
int main(){
vector<string>words{"abc","deq","mee","aqq","dkd","ccc"};
string pattern = "xyz";
vector<string>finalAns= findSimilarPattern(words,pattern);
for(int i=0; i< finalAns.size(); i++){
    cout << finalAns[i] << " ";
}
}
```

Time complexity → O(n)

Space complexity → O(1) + O(n) → O(n)
O(1) → constant space for char array mapping
O(n) → ans vector